December 21, 2018

# ON-DEVICE ANALYTICS: DATA COLLECTION, REPRESENTATION, AND FEEDBACK LOOP

Vishal Murgai

Nikul Bhatt

Srinivas Addanki

# ON-DEVICE ANALYTICS: DATA COLLECTION, REPRESENTATION, AND FEEDBACK LOOP

AUTHORS:
Vishal Murgai
Nikul Bhatt
Srinivas Addanki

## ABSTRACT

Described herein are techniques for a Machine Learning (ML) model to learn from a training set and predict the cause of a failure accurately. Although it is challenging for a human to identify the root cause of an issue when configuration and show command outputs are very large (e.g., for thousands of Internet Protocol (IP) routes), troubleshooting on a network device typically has to rely on a manual process with the help of some show commands.

## DETAILED DESCRIPTION

On-device analytics for troubleshooting and providing assurance to a user requires data that can be understood by various Machine Learning (ML) models. Currently there is no structured data available for any of the network events which can be used as a training set for ML algorithms to consume. Log messages from network devices are totally unstructured data, and difficult to parse/analyze. All ML algorithms expect data in numerical format. Only after raw data has been converted to a structured/normalized/numerical format can ML algorithms use it for various purposes.

Troubleshooting is mostly manual today, and as such needs to be repeated every time leading to extremely high operating costs. The techniques described herein address the first step towards automation (troubleshooting / assurance / machine reasoning), i.e. generating a training set for network devices.

A combination of user intent and associated operational states may be used to collect data. A custom parser then reads through the collected data, normalizes it, and represents the data in a ML algorithm friendly format.

A single event at a time may be caused on the network device. The event may be a problematic event, and may be simulated via programmatic telnet access to the network device. A change may be injected using a Command Line Interface (CLI) or through a

neighbor (peer) device triggered event. An exact identification of the change may be provided as a label.

The operational data state caused by the single event may be collected based on a system snapshot using the CLI command 'show system snapshots dump <snapshot_name>'. Interest in data may differ between two snapshots. Differential information may be captured by comparing the problematic snapshot with a golden snapshot. This may be accomplished using the CLI command 'show system snapshots compare <snapshot-1> <snapshot-2>'. This may generate differences as shown in Figure 1 below.

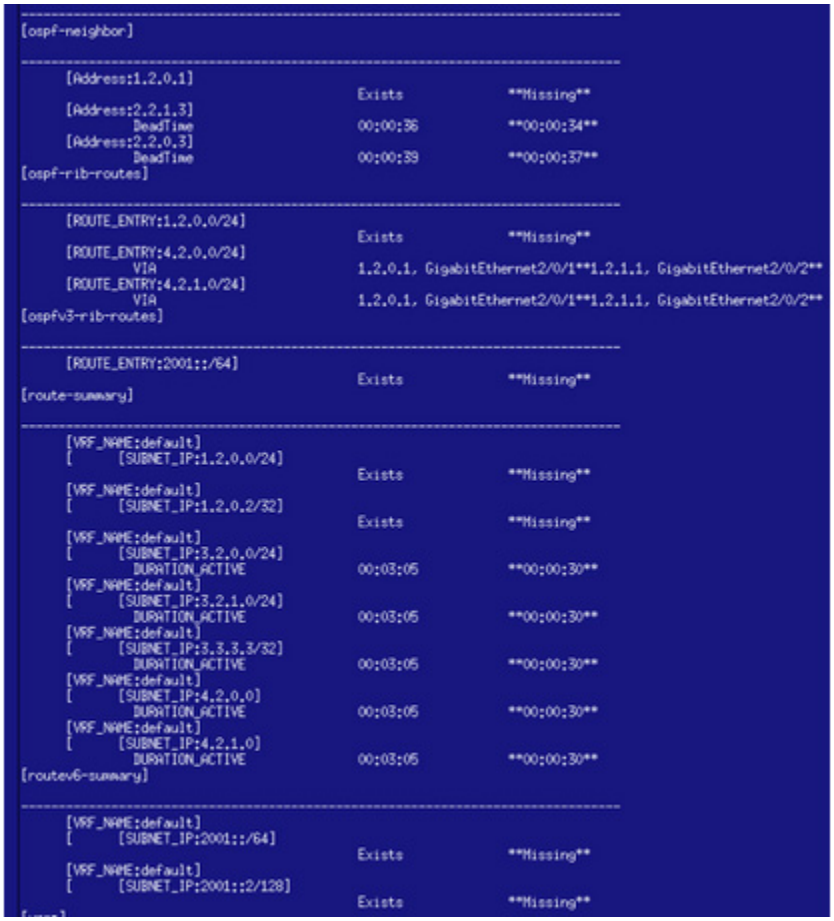


*Figure 1*

Many differential data sets may be generated with some of the most prominent failure conditions to train the model. A framework to automate the process of generating differentials may be utilized.

The results from the operational data may be stored in the form of a hierarchical dictionary. The differentials may be traversed line-by-line. The first component for which

changes are seen may be identified, and then attributes (under the component) which have changed may be identified. At the end of the parsing, a hierarchical dictionary is created for each component in the snapshot differentials. As illustrated in Figure 2 below, the component (e.g., root of the tree, such as "interface" in Figure 2) has entries for multiple attributes (which have their state changed since the golden snapshot).

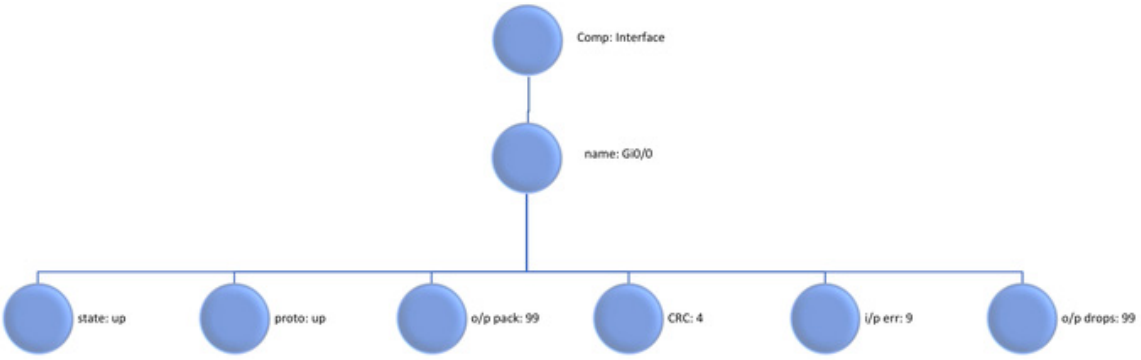


*Figure 2*

The snapshot differentials are parsed. The changed attributes are marked under a component as "1" and the unchanged attributes are marked under a component as "0". An example dictionary is shown in Figure 3 below.

| config_change | int_up | int_down | line_proto_up | ..... | issue (Label) |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | ..... | int_state_change |

*Figure 3*

The dictionary is traversed and normalized based on a set of criteria. The dictionary may be exported as a row into a CSV file, as illustrated in Figure 4 below. This custom CSV file may serve as training data to an ML algorithm.

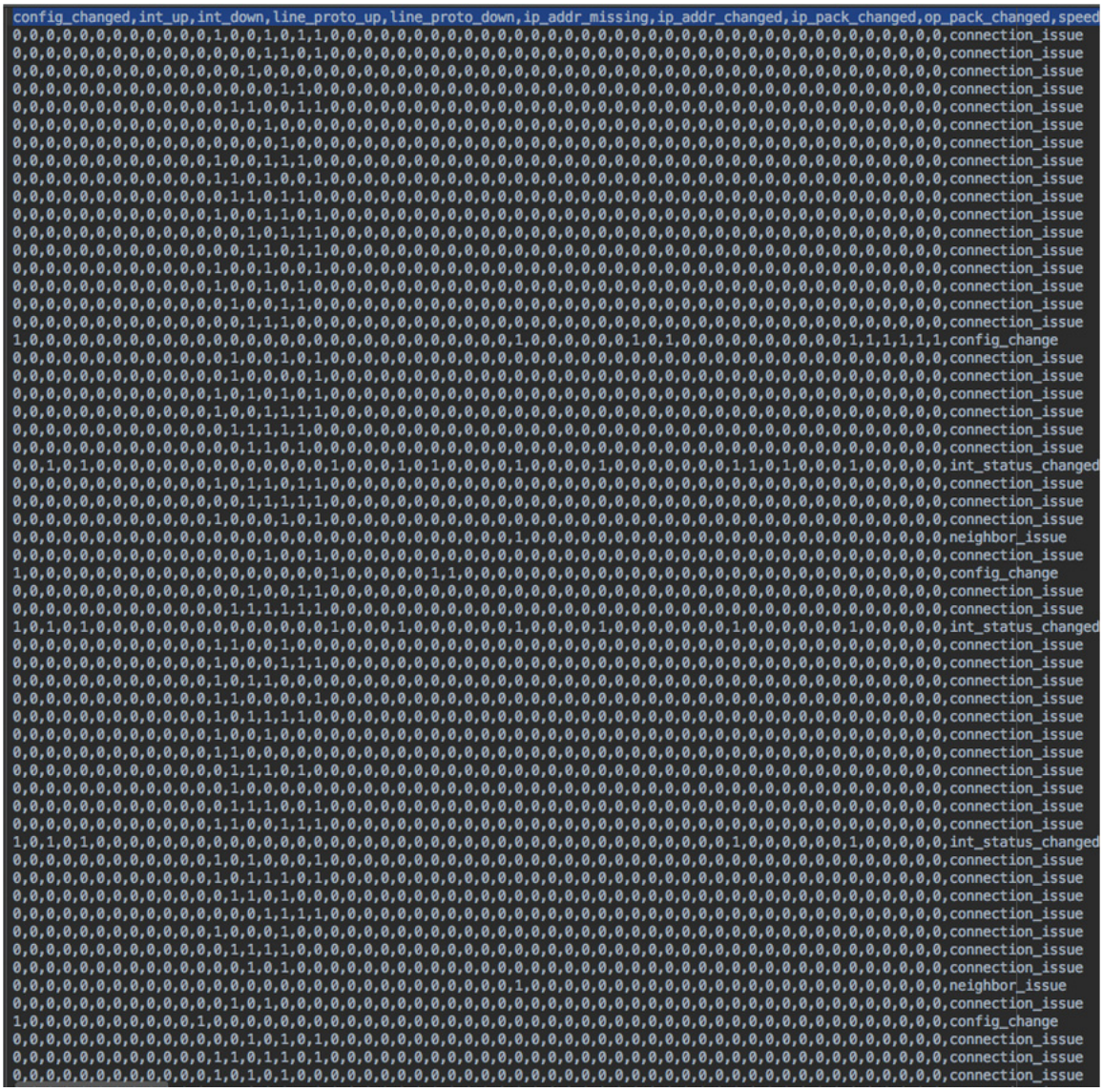


*Figure 4*

This may be repeated for N different network events to build a comprehensive training set, which may be fed into ML algorithms.

Figure 5 below illustrates a simple decision tree for predicting the cause of a state change.

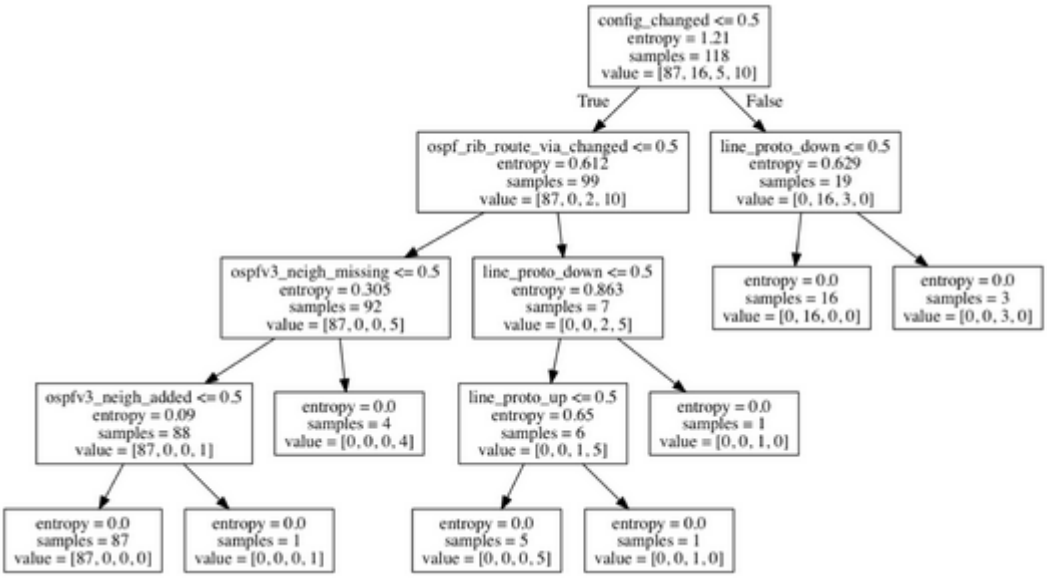


*Figure 5*

After the ML model has identified an issue, a mechanism may be built around to auto-correct the error introducing the event and have a complete feedback loop.

The techniques described herein may use a powerful combination of user intent (i.e., what the user wanted to do through configuration) and the system snapshot (i.e., how they were translated into the current working behavior of the network device). The differential of the current system configuration and a "golden configuration" are obtained along with differentials of a current system snapshot (reflecting the current system state) and a "golden snapshot". The differentials may be binary encoded and fed to a trained ML model. The model then predicts the possible event which might have caused the issue.

These mechanisms may convert differential information (between two states of a network device) into an organized binary data set which can be fed into a suitable ML algorithm to predict failure in a network device. Furthermore, network device behavior may be corrected by undoing the changes which caused the failure.

In summary, described herein are techniques for a ML model to learn from a training set and predict the cause of a failure accurately. Although it is challenging for a human to identify the root cause of an issue when configuration and show command outputs are very large (e.g., for thousands of Internet Protocol (IP) routes), troubleshooting on a network device typically has to rely on a manual process with the help of some show commands.